Alignment & Fine-tuning

Lecture 16 · ES 667: Deep Learning

Prof. Nipun Batra
IIT Gandhinagar · Aug 2026

Learning outcomes

By the end of this lecture you will be able to:

1. Explain why pretrained LLMs are **not chat assistants** out of the box.
2. Write the **instruction-tuning** recipe (SFT on demo data).
3. Implement **LoRA** adapter in ~20 lines of PyTorch.
4. Compute **LoRA parameter count** and memory savings for a 70B model.
5. Derive the **DPO loss** from the RLHF optimal-policy closed-form.
6. Pick **SFT / LoRA / QLoRA / RLHF / DPO** for a given alignment task.

Where we are

- **Pretrained LLMs** (L15) · Chinchilla-optimal, RoPE, GQA — raw capability.
- But raw pretrained models are **not** chat assistants. Raw GPT-3 will complete your prompt with plausible internet text, not with a useful answer.

> REFERENCE
>
> Today maps to HF PEFT docs + the **InstructGPT paper** (Ouyang 2022) + **DPO paper** (Rafailov 2023).

Four questions:

1. How do we turn pretrained models into **instruction followers**?
2. What is **LoRA** and why did it eat the fine-tuning world?
3. How does **RLHF** differ from **DPO**?
4. What's happening with **reasoning models** (o1, Claude thinking)?

PART 1

SFT · Instruction tuning

The first step after pretraining

Why SFT is necessary

A pretrained model sees:

> *"The capital of France is"* → *completes with "Paris"*

Not bad. But give it:

> *"What is the capital of France?"*

And it might reply with "What is the capital of Italy? What is the capital of Spain? ..." — it treats your question as a prompt to continue generating FAQ-style content.

SFT fixes this. Train on `(instruction, response)` pairs — teach the model *what a helpful response looks like*.

SFT in practice

```
# A typical SFT training example
conversation = [
    {"role": "system",    "content": "You are a helpful assistant."},
    {"role": "user",      "content": "What is the capital of France?"},
    {"role": "assistant", "content": "The capital of France is Paris."}
]

# Apply chat template → single string with delimiters
text = tokenizer.apply_chat_template(conversation)

# Train with standard causal LM loss, but **only on the assistant response**
# (mask system + user tokens)
loss = ce_loss(logits, labels, label_smoothing=0.0)
```

Dataset sizes: 10k–1M high-quality examples. Much smaller than pretraining (trillions). But the effect is massive.

SFT gotchas

WATCH OUT

Overtraining — SFT on too narrow a dataset makes the model parroting rather than helpful.

Forgetting — aggressive SFT can destroy pretraining knowledge. Use small LRs (1e-6 to 1e-5).

Loss masking — if you don't mask non-response tokens, the model "learns" to produce the user's question.

PART 2

LoRA · parameter-efficient fine-tuning

The technique everyone uses

Why full fine-tuning is painful

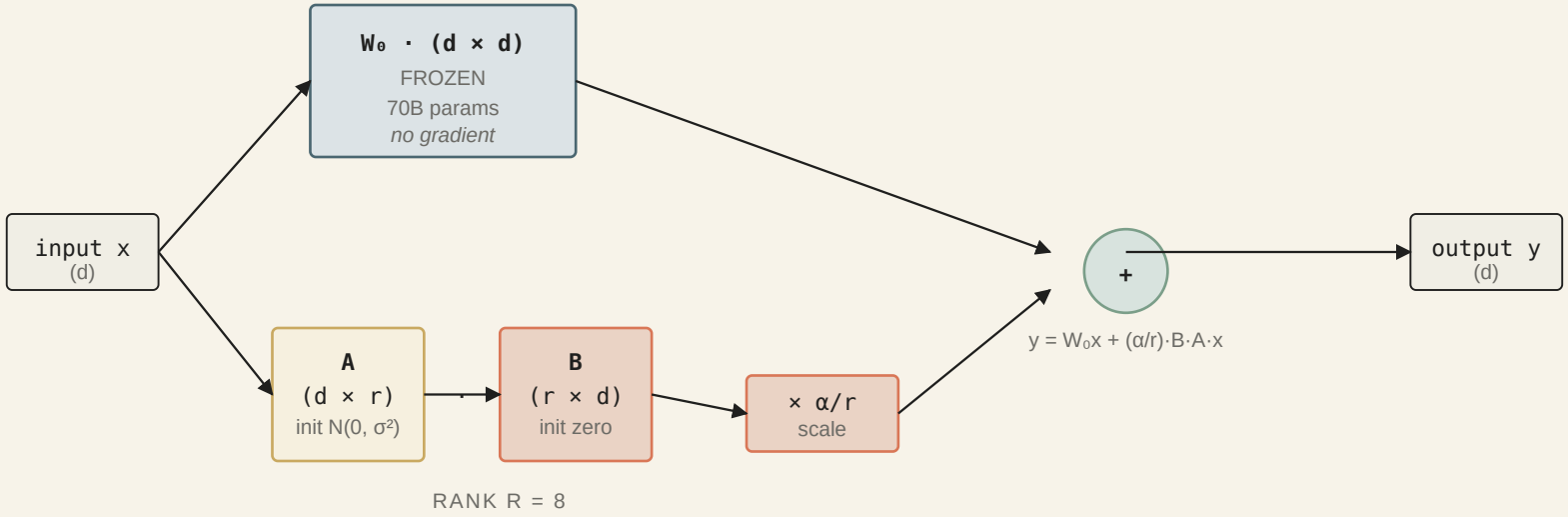
To fine-tune a 70B model with Adam:

- **Weights** · 70B × 2 bytes (bf16) = 140 GB
- **Gradients** · 140 GB
- **Optimizer state** (Adam m and v) · 2 × 280 GB
- **Activations** · ~200 GB (depends on batch)

Total: **~760 GB**. Needs an H100 cluster. Most people don't have that.

LoRA · the 2021 fix

LoRA — train a low-rank update alongside a frozen base matrix

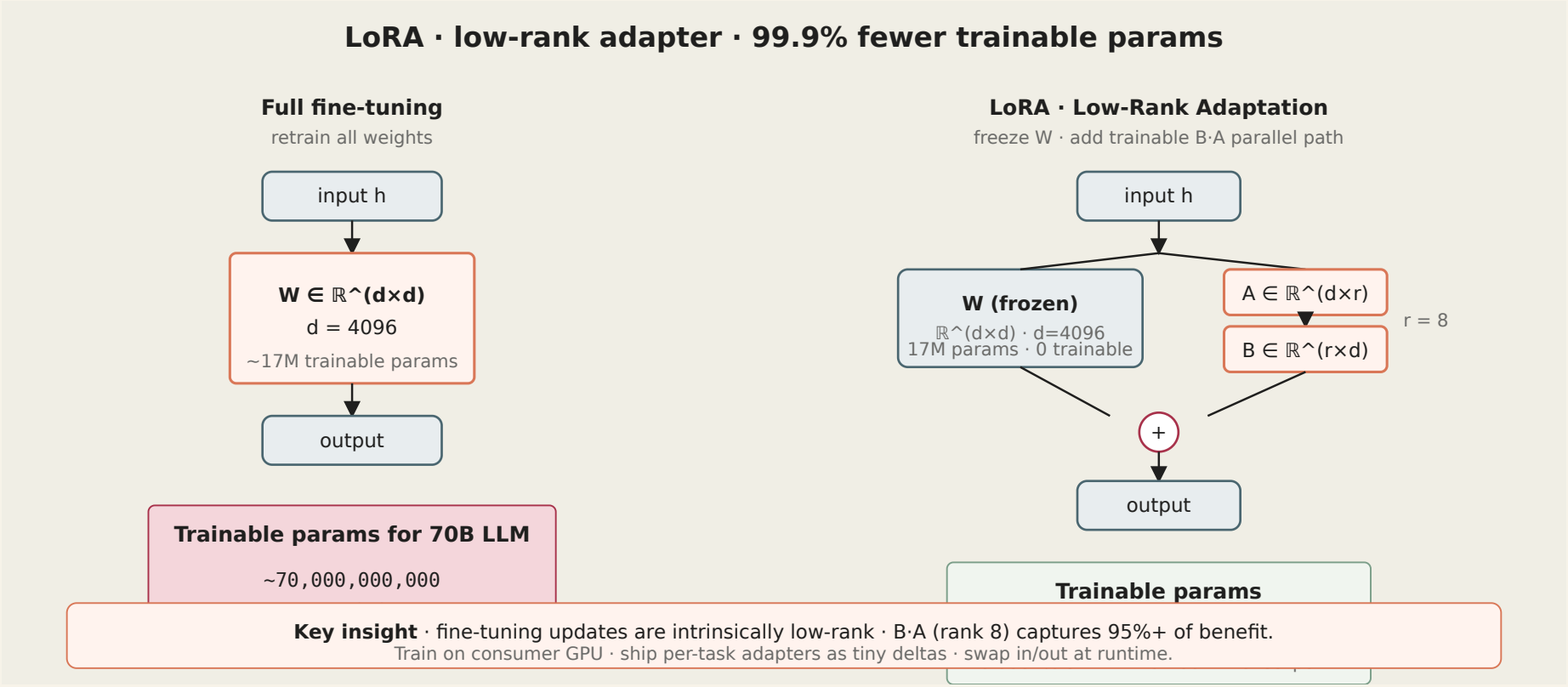


PARAMETER COUNT

Base W_0 · d^2 = 70B (frozen) · LoRA A, B · $2 \cdot d \cdot r$ = **0.1–1% of base**

With r = 8, d = 4096: W_0 has 16M params; A·B has 64k · 500× smaller to train, 500× smaller to save as adapter.

LoRA · detailed view



IN PRACTICE

▶

Interactive: slide the rank, see parameter counts drop 100×–1000× — lora-adapter.

LoRA · the oil-painting analogy

KEY IDEA

A pretrained model is a **giant oil painting**. Fine-tuning for a new style — say, making the subject smile — doesn't require **repainting the canvas**. A few brushstrokes in the right places do the job.

LoRA finds *those* brushstrokes · two small low-rank matrices that adjust the original weights · the masterpiece beneath stays untouched.

That's why a 70B-param fine-tune can ship as a 100MB adapter file · the brushstroke layer · while the base painting stays the same.

LoRA · build the update from a tiny rank-1 example

Pretrained W_0 is 4×4 (16 params, frozen). Full update ΔW would also have 16 params.

LoRA's idea · construct ΔW from two rank-1 matrices.

- $A \in \mathbb{R}^{4\times 1}$ · 4 trainable params.
- $B \in \mathbb{R}^{1\times 4}$ · 4 trainable params.
- $\Delta W = BA$ is 4×4 but generated from only **8** numbers.

$$\Delta W = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} \begin{pmatrix} b_1 & b_2 & b_3 & b_4 \end{pmatrix} = \begin{pmatrix} a_1 b_1 & a_1 b_2 & a_1 b_3 & a_1 b_4 \\ a_2 b_1 & a_2 b_2 & a_2 b_3 & a_2 b_4 \\ a_3 b_1 & a_3 b_2 & a_3 b_3 & a_3 b_4 \\ a_4 b_1 & a_4 b_2 & a_4 b_3 & a_4 b_4 \end{pmatrix}$$

Train only A, B; keep W_0 frozen:

$$W = W_0 + \frac{\alpha}{r} BA$$

At $t = 0$, $B = 0 \Rightarrow BA = 0$ → output equals base model.

Worked numeric · LoRA on a Llama-7B layer

Typical layer · $d = 4096$, weight 4096×4096.

Full fine-tune. Trainable = $4096^2 = \mathbf{16{,}777{,}216}$ per layer.

LoRA at rank $r = 8$.

- A: $4096 \times 8 = 32{,}768$ params.
- B: $8 \times 4096 = 32{,}768$ params.
- Total · $32{,}768 + 32{,}768 = \mathbf{65{,}536}$.

  Savings · $16{,}777{,}216 / 65{,}536 = \mathbf{256}\times$ **fewer params per layer.** That's why a 70B fine-tune can ship as a 100 MB adapter — only A and B for each adapted layer, not the base.

LoRA numbers · 7B model

METHOD	TRAINABLE PARAMS	RATIO	DISK
Full fine-tune	7B	100%	14 GB
LoRA · r=8	~4M	0.06%	8 MB
LoRA · r=64	~33M	0.47%	66 MB
QLoRA · 4-bit base + r=8	~4M	0.06%	8 MB + 3.5 GB base

IN PRACTICE

Ship the 8 MB adapter alongside the public 7B base. Everyone downloads the base once; each task is just an adapter swap. This changed the open-source LLM ecosystem.

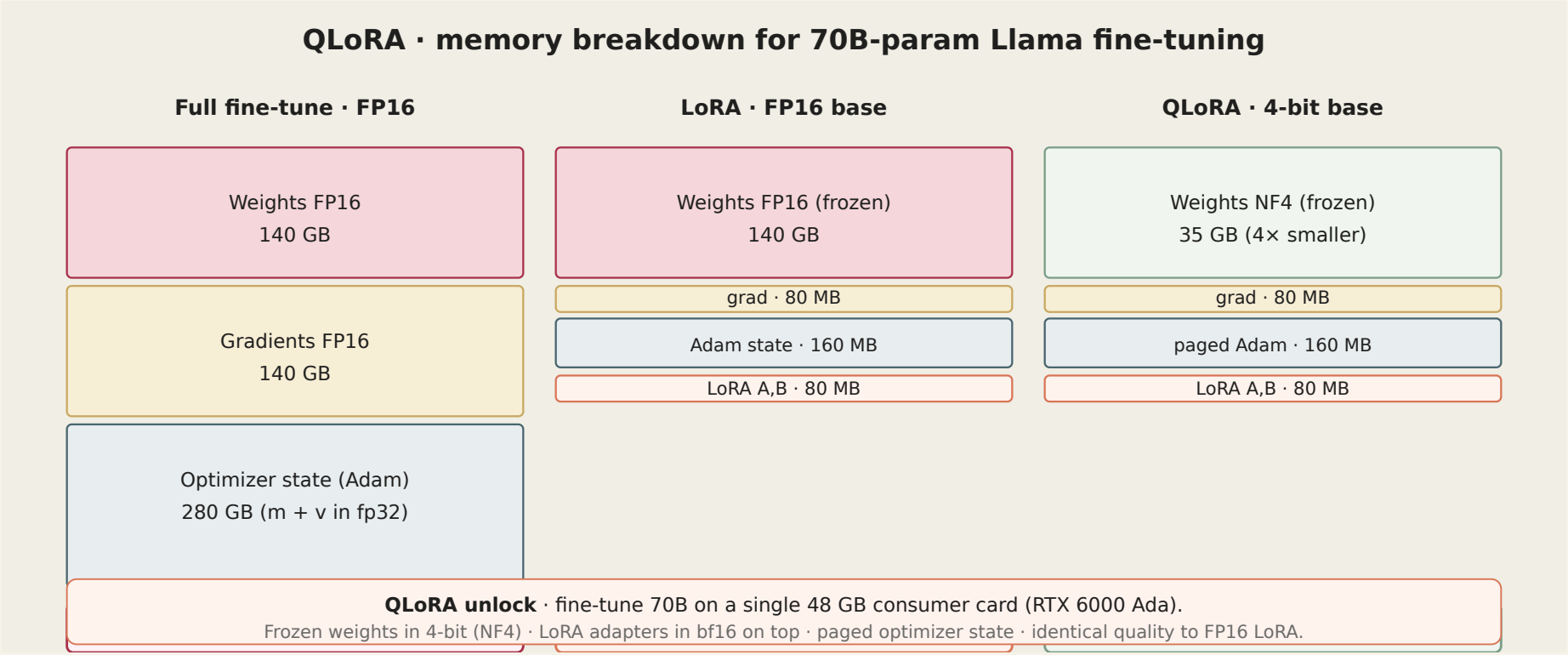
LoRA in PyTorch · peft library

```python
from peft import LoraConfig, get_peft_model

lora_cfg = LoraConfig(
    r=8,
    lora_alpha=16,
    target_modules=["q_proj", "v_proj"],  # where to inject
    lora_dropout=0.05,
    bias="none",
)
model = get_peft_model(base_model, lora_cfg)
model.print_trainable_parameters()
# trainable params: 4,194,304  all: 7,000,000,000  trainable%: 0.06%
```

Training loop is identical to normal SFT — only 0.06% of parameters have gradients, everything else is frozen.

QLoRA · memory breakdown



QLoRA · quantization is image-compression

INTUITION

Analogy. A photo `.bmp` (millions of colours) → save as `.gif` (256-colour palette). The GIF maps each pixel to its closest palette colour. Looks similar, vastly smaller.

QLoRA does this to weights. **bf16** has $2^{16} = 65{,}536$ possible values; **4-bit** has only $2^4 = 16$. Map each weight to its closest 4-bit codebook value → 4× memory shrink.

NF4 ("NormalFloat 4-bit") · a clever choice of those 16 codebook values, spaced to match the typical bell-shaped distribution of NN weights.

QLoRA · worked memory budget for 70B

Per parameter:

- bf16 → 2 bytes
- 4-bit → 0.5 bytes

Standard LoRA (bf16 base).
$70 \times 10^9 \cdot 2 = 1.4 \times 10^{11}$ bytes = **140 GB** → needs 2× A100 80 GB.

QLoRA (4-bit base).
$70 \times 10^9 \cdot 0.5 = 3.5 \times 10^{10}$ bytes = **35 GB** → fits on a single A100 40 GB or a consumer 48 GB GPU. LoRA adapters add < 100 MB.

Mechanism.

1. Load base, quantize to 4-bit (4× shrink).
2. Freeze 4-bit weights.
3. Attach **bf16** LoRA adapters. Train only these.
4. During the forward pass, dequantize chunks of W_0 to bf16 for each matmul, then discard.

Used everywhere in open-source LLM fine-tuning since 2023.

PART 3

RLHF · reward model + PPO

The alignment loop that made ChatGPT

Why SFT isn't enough

SFT teaches the model *one way* to respond to each prompt. But for open-ended questions, many responses are acceptable — and you want the model to prefer the *best* one.

> *"Explain quantum entanglement to a 12-year-old."*

- Response A · accurate, clear, uses an analogy. ✓
- Response B · accurate but terse and dry. ≈
- Response C · inaccurate, overly technical. ✗

SFT would weight all three equally if all are in the dataset. **RLHF** adds preference learning.

SFT vs RLHF · dog-training analogy

KEY IDEA

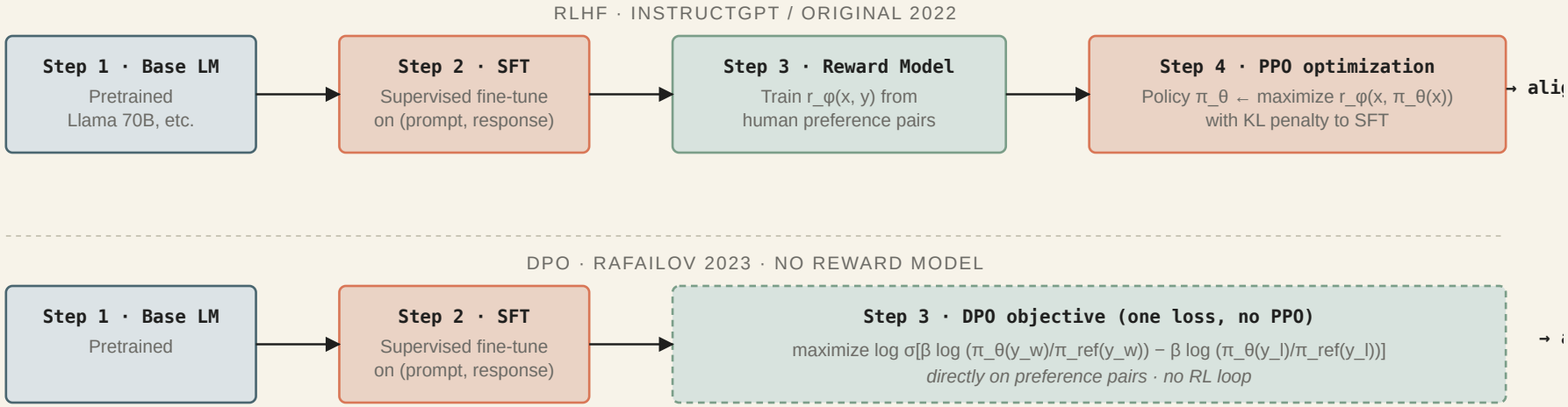
SFT is teaching a dog the command "sit." One correct action; demonstrate, repeat.

RLHF is teaching a dog to choose between options · "bark or stay quiet when the doorbell rings" · we **reward** good choices, **penalize** bad ones.

By showing the model pairs of options (good answer · bad answer) and letting it learn from preference, we shape *general behavior* rather than memorize a single correct response.

The RLHF pipeline

RLHF vs DPO — both align a pretrained model, one adds a reward model step, one doesn't



WHY DPO REPLACED MOST RLHF BY 2024

- One stage instead of two (no reward model to train, debug, balance)
- Simpler code — pure supervised loss, no PPO
- Often matches RLHF quality on benchmarks; still debated for top-tier quality
- Anthropic / OpenAI / Google still use RLHF variants at the frontier
- Open-source (Mistral, Hugging Face) has converged on DPO
- *2025 variants: IPO, KTO, SimPO — all refinements of DPO*

RLHF · the dog-with-two-goals analogy

INTUITION

Train a dog to fetch.

1. **Get the ball** → treat. The dog maximizes treats. (**Reward** r_ϕ.)
2. **Don't go crazy** → if it knocks over furniture chasing the ball, you tug a leash to keep it close to its normal behaviour. (**KL penalty** D_{KL}.)

RLHF balances both · max reward, but stay near the SFT model.

RLHF objective · term by term

$$\max_{\theta} \underbrace{\mathbb{E}_{x\sim D}\big[\, r_\phi(x, \pi_\theta(x))\,\big]}_{\text{Part 1: maximize reward}} - \underbrace{\beta\, D_{\mathrm{KL}}(\pi_\theta \| \pi_{\mathrm{ref}})}_{\text{Part 2: stay near SFT}}$$

Symbols:

- π_θ · trainable model ("policy"); takes prompt x, generates response y.
- π_{ref} · frozen SFT model.
- $r_\phi(x, y)$ · reward model; scores responses (e.g. 0.85).
- $\mathbb{E}_{x\sim D}[\cdot]$ · average over prompts in dataset D.
- $D_{\mathrm{KL}}(\pi_\theta \,\|\, \pi_{\mathrm{ref}})$ · KL divergence; high when π_θ diverges from π_{ref}.
- β · leash strength.

Part 1 drives the model to produce high-reward answers. **Part 2** keeps the policy close to its SFT initialization → prevents the dog from tearing up the garden.

Worked example · one RLHF step

Prompt. "Explain gravity to a 5-year-old."

- π_{ref} would say *"Gravity is a fundamental interaction..."* — assigns this prob 0.001 (technical, not for kids).
- π_θ generates *"Imagine the earth is a big magnet and you have tiny magnets in your shoes..."* — assigns prob 0.2.

Reward. $r_\phi \approx 0.95$ — friendly, age-appropriate.

KL penalty. Contribution from this example $\propto \log(0.2/0.001) = \log 200 \approx 5.3$ — sizeable.

Total objective. $0.95 - \beta \cdot 5.3$.

PPO updates θ to push *toward* this kind of response (high reward) while not pulling too far from π_{ref} (KL penalty). Without KL, the model would happily reward-hack into nonsense; the leash holds it close.

RLHF · what could go wrong

WATCH OUT

Reward hacking. The policy finds ways to game the RM that humans wouldn't approve of. Classic example: the RM rewards longer answers, so the policy learns to produce verbose output regardless of content.

Mode collapse. PPO can push the policy to always produce very similar responses — diversity loss.

Sycophancy. If human labelers preferred agreeable answers, the model learns to agree with whatever the user says, even when wrong.

Mitigating these is half the art of alignment.

PART 4

DPO · Direct Preference Optimization

Bypass the reward model entirely

DPO · the 2023 simplification

REFERENCE

Rafailov et al. 2023 · *"Direct Preference Optimization: Your Language Model is Secretly a Reward Model."*

Insight: the RLHF objective has a closed-form optimal policy given the reward. Work backwards — derive a loss directly on preference pairs, skipping the RM and PPO entirely.

DPO · the simpler taste-test analogy

INTUITION

RLHF · two sodas → judge scores each 1–10 → use scores to declare a winner. Indirect.
DPO · just ask "A or B?" → direct preference.

DPO writes a loss that says · *"directly increase prob of the winner y_w, decrease prob of the loser y_l, relative to the SFT baseline."*

DPO loss · inside-out

$$\mathcal{L}_{\text{DPO}} = -\log \sigma \left(\underbrace{\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)}}_{\text{winner score}} - \underbrace{\beta \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}}_{\text{loser score}} \right)$$

Build it from inside:

1. **Ratio** $\pi_\theta / \pi_{\text{ref}}$ · how much more likely is the new model to produce y than the SFT model? Want > 1 for y_w.
2. **Log ratio** · "improvement score." Positive = improved over SFT.
3. **Difference** · winner score − loser score. Want **large positive**.
4. **Sigmoid** · squash to $(0, 1)$. Large positive diff → $\sigma \approx 1$.
5. $-\log \sigma(\cdot)$ · standard CE loss. $\sigma \to 1 \Rightarrow$ loss $\to 0$. $\sigma \to 0 \Rightarrow$ huge loss → big gradient kicks in.

Worked numeric · DPO step

Prompt · "Suggest a coffee shop name." Winner y_w = "The Daily Grind". Loser y_l = "Coffee Shop". $\beta = 1$.

	π_{ref}	π_θ
y_w	0.05	0.06
y_l	0.10	0.12

Winner score. $\log(0.06/0.05) = \log 1.2 \approx 0.18$.
Loser score. $\log(0.12/0.10) = \log 1.2 \approx 0.18$.

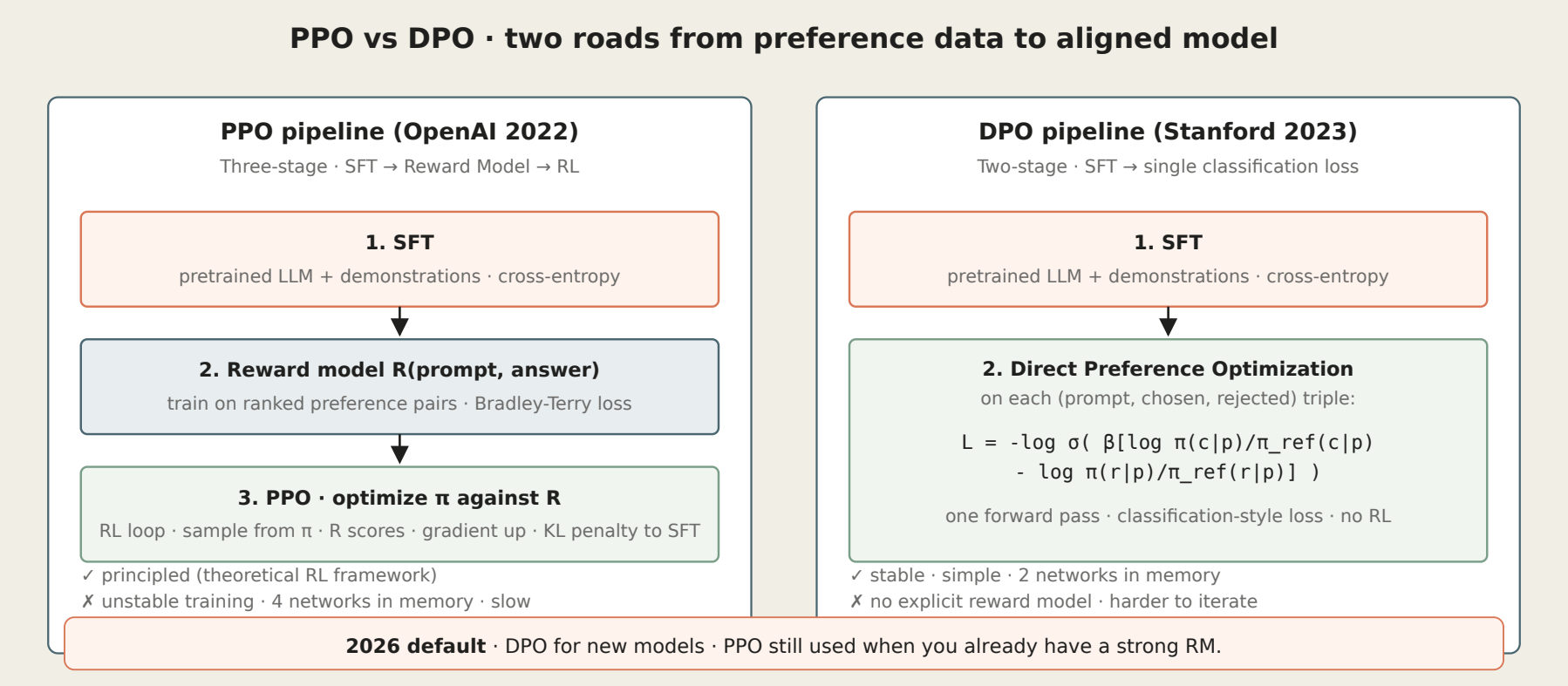
Difference = $0.18 - 0.18 = \mathbf{0}$. Both got the same boost — model hasn't learned the preference yet.

Loss = $-\log \sigma(0) = -\log 0.5 \approx \mathbf{0.69}$ — non-zero.

Gradient pushes θ to increase $\pi_\theta(y_w)$ (e.g. to 0.08) and decrease $\pi_\theta(y_l)$ (e.g. to 0.11). Now the difference is positive → loss drops.

Pure supervised loss. No RL loop. No reward model. ~50 lines vs RLHF's thousands.

PPO vs DPO · the two pipelines



DPO vs RLHF · which to use

	DPO	RLHF
Training stages	1	2 (RM + PPO)
Code complexity	~50 LoC	~2000 LoC (PPO, RM, rollout)
Compute	1×	3–5×
Quality at top scale	tied	often slightly ahead
Open-source preference	**DPO**	RLHF for frontier labs

IN PRACTICE

Open-source (Hugging Face, Mistral, most Llama fine-tunes) · **DPO**. Frontier labs (Anthropic, OpenAI, Google) · **RLHF variants** with proprietary tooling. Both work.

Constitutional AI and RLAIF

Anthropic 2022 variation · use the model itself (or a stronger one) to generate preference labels from a written "constitution":

1. Draft a response.
2. Ask the model: *"Does this violate principle X?"*
3. Revise the response.
4. Use revisions as preference pairs, train with RLAIF (RLHF with AI feedback).

Scales human annotation by factors of 100+. Used in Claude's alignment pipeline.

PART 5

Reasoning models · the 2024 turn

Test-time compute as a new axis

Reasoning models

Latest generation · **o1, o3, Claude extended thinking, DeepSeek R1.**

The core idea:

1. Train the model to produce **long internal chain-of-thought** before responding.
2. Use RL with **process rewards** (reward good reasoning steps, not just final answer).
3. At inference, let the model think for minutes — spend 10×–100× more compute.

Result: dramatically better on math, code, logic benchmarks.

> INTUITION
>
> Scaling laws in training compute produced pretrained capability. A new axis — **scaling test-time compute** — now unlocks reasoning. Both will likely continue.

Process rewards · what "reasoning training" looks like

Traditional RL reward · "was the final answer correct?" · sparse, late signal.

KEY IDEA

Process reward model (PRM) · grades individual reasoning steps · dense signal, catches bad intermediate logic.

DERIVATION

STAGE	INPUT	OUTPUT
PRM training	100k human-labeled step-by-step proofs	reward-per-step model
RL with PRM	sampled chains-of-thought	step-reward gradient up
Result	chains improve *step-by-step*, not just final	better generalization

OpenAI's "math-shepherd" (2024) · PRM-trained models beat outcome-reward-only models by 20+ points on AIME. Process > outcome rewards for multi-step reasoning.

Reasoning models · benchmark jump

DERIVATION

MODEL	AIME 2024 (MATH)	CODEFORCES ELO	GPQA (SCIENCE)
GPT-4	12%	~800	39%
o1-preview	44%	~1500	73%
o1	74%	~1900	78%
o3	**97%**	**~2700** (grandmaster)	**88%**

INTUITION

o3 at 97% on AIME · humans gold-medal at ~85%. **One year** of inference-compute scaling delivered this jump. Same base model class; the training regime changed.

Picking an alignment method

DERIVATION

SCENARIO	RECOMMENDED
Small task-specific dataset (< 10k)	SFT on full model
Large instruction dataset (100k+)	SFT + LoRA
Consumer GPU (≤ 48GB) on 70B model	QLoRA (NF4 + LoRA)
Need preference alignment, small team	DPO
Need precise control of behaviors	RLHF + custom RM
Need safety + low-cost labels	Constitutional AI / RLAIF
Need reasoning / math / code	SFT + RL-with-process-rewards (o1 style)

IN PRACTICE

In 2026 open source · QLoRA + DPO is the dominant recipe for instruction tuning. Frontier labs mix all of the above.

Summary · Lecture 16 — summary

- **SFT** · train on (instruction, response) pairs to turn a pretrained LM into an assistant.
- **LoRA** · fine-tune two small matrices alongside a frozen base · 100× fewer trainable params. **QLoRA** adds 4-bit quantization to cut memory 4×.
- **RLHF** · SFT → reward model → PPO against RM with KL penalty to SFT. Original ChatGPT recipe.
- **DPO** · closed-form alignment without an RM · one supervised loss. Default in open-source.
- **Constitutional AI / RLAIF** · self-critique to scale preference data.
- **Reasoning models** (2024+) · RL with process rewards · long internal chain of thought · new scaling axis.

Read before Lecture 17

Prince Ch 14 (unsupervised, contrastive).

Next lecture

Self-Supervised & Contrastive Learning — SimCLR, BYOL, MAE, DINOv2. How to learn without labels.

NOTEBOOK

Notebook 16 · `16-lora-finetune.ipynb` — fine-tune a 7B model with LoRA + peft on a small instruction dataset · compare responses before / after.